3주차 결과보고서

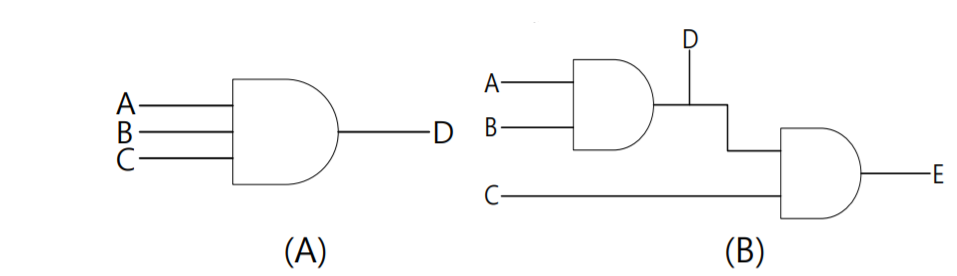
전공: 물리학과 학년: 3학년 학번: 20191286 이름: 김나현

1. FPGA 동작법을 설명하시오.

FPGA의 동작을 검증하기 위해서는 Verilog coding, Run synthesis, Device/Pin assignment, Synthesis/Implementation, Device configuration의 과정을 거쳐야 합니다.

본 실험에서는 3-input이나 4-input의 AND gate와 OR gate을 확인하는 것이 목적이기 때문에 3-input AND gate, 3-input OR gate, 4-input AND gate, 4-input OR gate, 즉 네 개의 verilog 코드를 작성합니다. 이때, Verilog 코드를 작성하고 왼쪽에 Project manager의 Settings라는 아이콘을 눌러 창이 열리면 General란에서 project device에 xc7a75tfgg484-1 (active)로 설정하고 확인을 눌러 device assignment를 합니다. Pin assignment를 위해, Project manager의 Add Sources를 눌러 창이 열리면 Add or create constraints를 누르고 next를 누릅니다. 그 다음에 뜨는 창에서 Create File을 눌러 inv이라는 이름으로 inv.xdc 파일을 생성합니다. inv.xdc 파일이 생성되면 FPGA pin list에서 할당하고 싶은 pin과 Verilog 소스의 port를 링크시키기 위해 해당 파일에 set\_property -dict {PACKAGE\_PIN G21 IOSTANDARD LVCMOS33} [get\_ports a] 또는 set\_property -dict {PACKAGE\_PIN F15 IOSTANDARD LVCMOS33} [get\_ports y]와 같이 코딩해줍니다. 이제 Verilog 창에 맨 위에 있는 Window라는 아이콘을 눌러 I/O Ports를 눌러 Fixed를 클릭해줍니다. Synthesis와 implementation 과정을 거치기 위해 Run synthesis와 Run implementation을 눌러주고 Generate Bitstream을 눌러 Bitstream 생성이 성공적으로 완료됐다는 창이 뜨면 open hardware manager을 클릭합니다. Program and debug>>open hardware manager>>open target>>auto connect를 누르고, project이름>>project이름\_runs>>impl\_1에 debug\_nets.lts 파일을 넣어줍니다. 다시 Verilog 창으로 돌아와, Program and debug>>open hardware manager>>program device>>xc7a75t\_0을 눌러 debug probes file을 project이름/project이름\_runs/impl\_1/debug\_nets.lts으로 설정하고 확인을 누른 후, FPGA 동작 결과를 확인합니다.

1. 3-input AND gate의 simulation 결과 및 과정에 대해서 설명하시오.



<Figure 1> 3-input AND gate를 구현하는 두 가지 방법

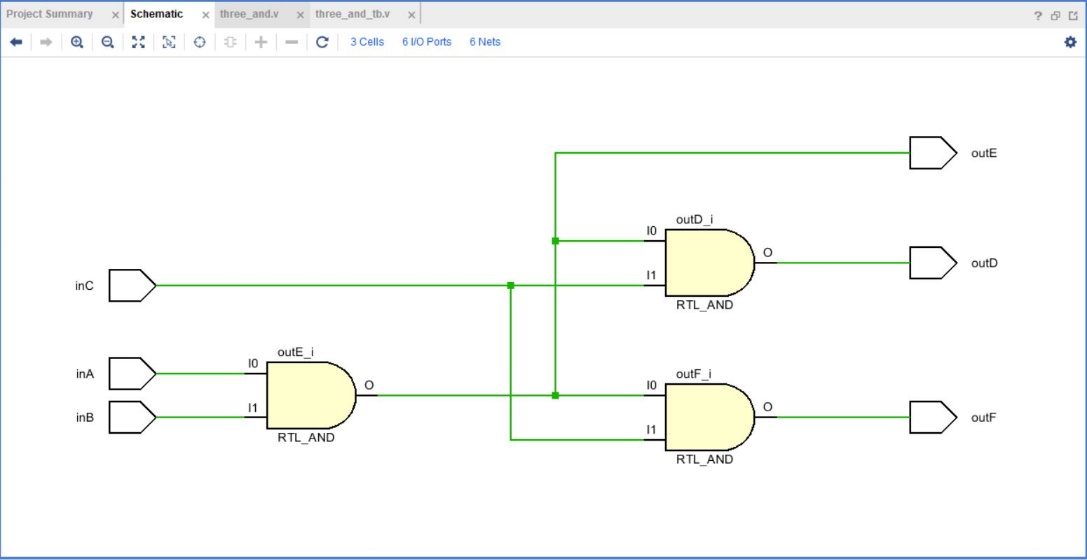
위의 <Figure 1>에서 (A)는 세 개의 입력이 모두 한 번에 하나의 AND gate에 인가되어 D라는 결과를 얻는 것이고, (B)는 AND gate에 A와 B, 두 개의 입력을 인가하여 D를 얻고, D와 C가 다른 AND gate의 입력으로 들어가 최종적으로 E라는 출력을 얻는 것입니다. 위에서 (A)의 출력 D는 outD로, (B)의 출력 D는 outE로, 출력 E는 outF로 설정하여 Verilog를 통해 구현해보면 다음 <Figure 2>와 같습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

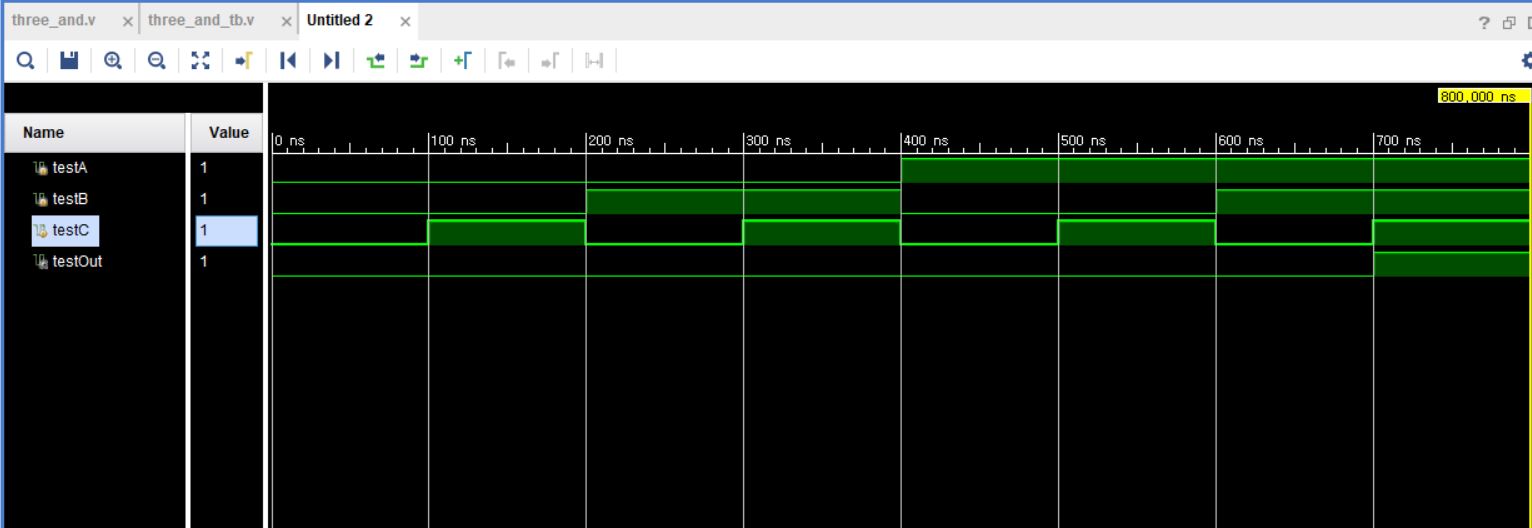
<Figure 2> three\_and.v 코드

<Figure 2>에 assign이 적힌 부분을 보면, outD는 inA와 inB, inC를 한 번에 이용하여 and 연산을 한 결과임을 알 수 있고, outE는 inA와 inB를, outF는 outE와 inC, 두 개의 입력을 각각 and 연산한 결과임을 알 수 있습니다. 이 코드의 schematic diagram을 확인해보면 다음 <Figure 3>와 같습니다.



<Figure 3> 3-input AND gate의 schematic diagram

위의 <Figure 3>를 보면, <Figure 1>에서 (A)와 (B)에서의 최종적인 출력, D와 E를 의미하는 outD를 얻는 과정이나 outF를 얻는 과정이 다르지 않음을 알 수 있습니다. <Figure 1>의 (A)처럼 3개의 입력이 하나의 AND gate로 인가되어 하나의 출력을 얻는다고 생각하고 <Figure 2>처럼 코드를 짜도 결국 <Figure 1>의 (B)와 동일하게 AND gate를 두 번 거치는 과정으로 구현된다는 사실을 알 수 있습니다.



<Figure 4> 3-input AND gate의 simulation 결과

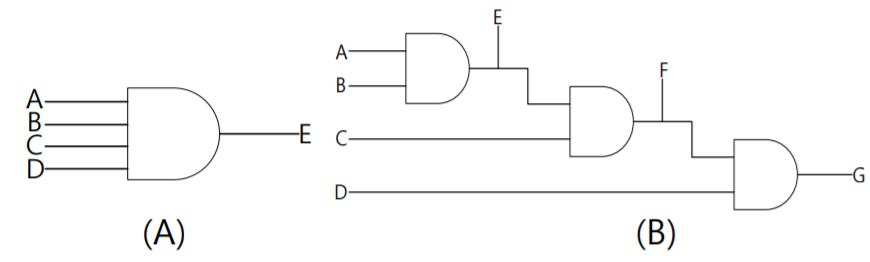
위의 <Figure 4>는 <Figure 3>을 통해 <Figure 1>의 (A)와 (B)는 동일한 출력 결과를 갖는다는 것을 확인한 후, test bench 코드를 이용하여 A, B, C라는 입력을 시간에 따라 바꿔가며 최종적인 출력 결과의 변화를 확인한 simulation 결과입니다 이 simulation 결과를 이용하여 3-input AND gate의 진리표를 작성해보면 다음 <Table 1>과 같습니다.

<Table 1> 3-input AND gate의 진리표

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | | | Output |
| A | B | C | D |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

AND 게이트는 모든 입력이 1일 때만 1을 출력하므로 위의 <Figure 4>의 simulation 결과가 정확하게 나타난 것임을 알 수 있습니다. 이때, <Table 1>의 output D는 <Figure 1>의 (A)의 D와 (B)의 E를 의미합니다.

1. 4-input AND gate의 simulation 결과 및 과정에 대해서 설명하시오.



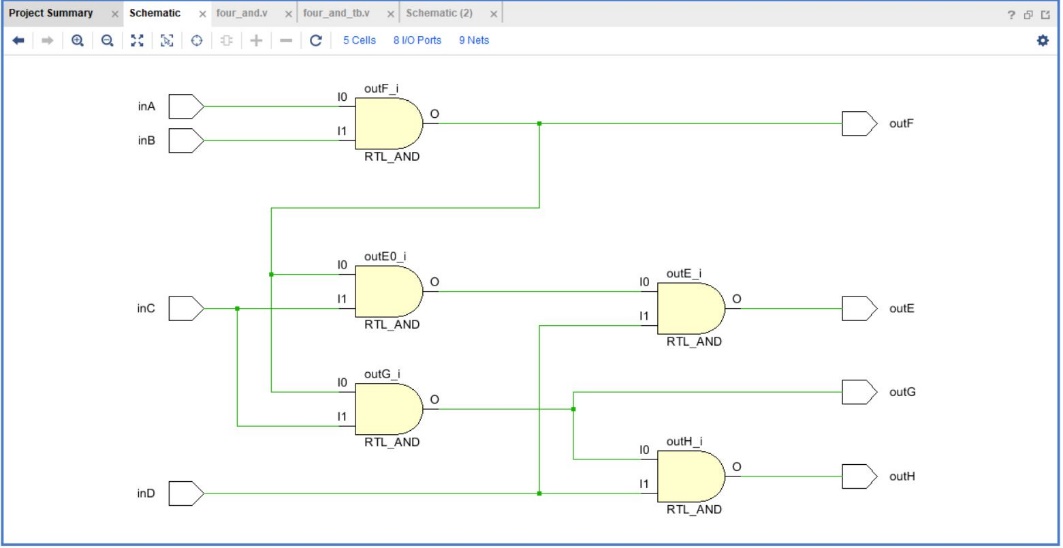
<Figure 5> 4-input AND gate를 구현하는 두 가지 방법

텍스트이(가) 표시된 사진

자동 생성된 설명위의 <Figure 5>에서 (A)는 네 개의 입력이 모두 한 번에 하나의 AND gate에 인가되어 E라는 결과를 얻는 것이고, (B)는 AND gate에 A와 B, 두 개의 입력을 인가하여 E를 얻고, E와 C가 다른 AND gate의 입력으로 들어가 F를 얻고, F와 D가 다른 AND gate의 입력으로 들어가 최종적으로 E라는 출력을 얻는 것입니다. 위에서 (A)의 출력 E는 outE로, (B)의 출력 E는 outF로, 출력 F는 outG, 출력 G는 outH로 설정하여 Verilog를 통해 구현해보면 다음 <Figure 6>와 같습니다.

<Figure 6> four\_and.v 코드

<Figure 6>에 assign이 적힌 부분을 보면, outE는 inA와 inB, inC, inD를 한 번에 이용하여 and 연산을 한 결과임을 알 수 있고, outF는 inA와 inB를, outG는 outF와 inC를, outH는 outG와 inD를, 즉 두 개의 입력을 각각 and 연산한 결과임을 알 수 있습니다. 이 코드의 schematic diagram을 확인해보면 다음 <Figure 7>와 같습니다.



<Figure 7> 4-input AND gate의 schematic diagram

위의 <Figure 7>를 보면, <Figure 5>에서 (A)와 (B)에서의 최종적인 출력, E와 G를 의미하는 outE를 얻는 과정이나 outH를 얻는 과정이 다르지 않음을 알 수 있습니다. <Figure 5>의 (A)처럼 4개의 입력이 하나의 AND gate로 인가되어 하나의 출력을 얻는다고 생각하고 <Figure 6>처럼 코드를 짜도 결국 <Figure 5>의 (B)와 동일하게 AND gate를 세 번 거치는 과정으로 구현된다는 사실을 알 수 있습니다.



<Figure 8> 4-input AND gate의 simulation 결과

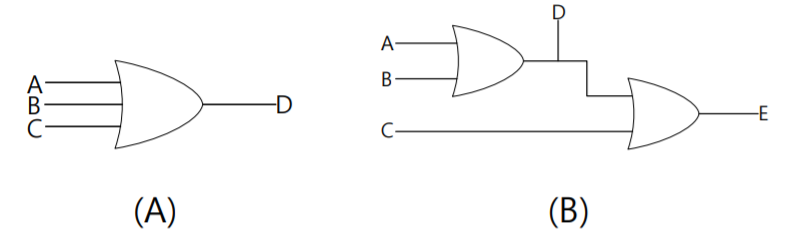
위의 <Figure 8>는 <Figure 7>을 통해 <Figure 5>의 (A)와 (B)는 동일한 출력 결과를 갖는다는 것을 확인한 후, test bench 코드를 이용하여 A, B, C, D라는 입력을 시간에 따라 바꿔가며 최종적인 출력 결과의 변화를 확인한 simulation 결과입니다 이 simulation 결과를 이용하여 4-input AND gate의 진리표를 작성해보면 다음 <Table 2>와 같습니다.

<Table 2> 4-input AND gate의 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inputs | | | | Output |
| A | B | C | D | E |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

<Table 2>를 보면, AND 게이트는 입력의 개수와 무관하게 모든 입력이 1일 때만 1을 출력하므로 위의 <Figure 8>의 simulation 결과가 정확하게 나타난 것임을 알 수 있습니다. 이때, <Table 2>의 output E는 <Figure 1>의 (A)의 E와 (B)의 G를 의미합니다.

1. 3-input OR gate의 simulation 결과 및 과정에 대해서 설명하시오.



<Figure 9> 3-input OR gate를 구현하는 두 가지 방법

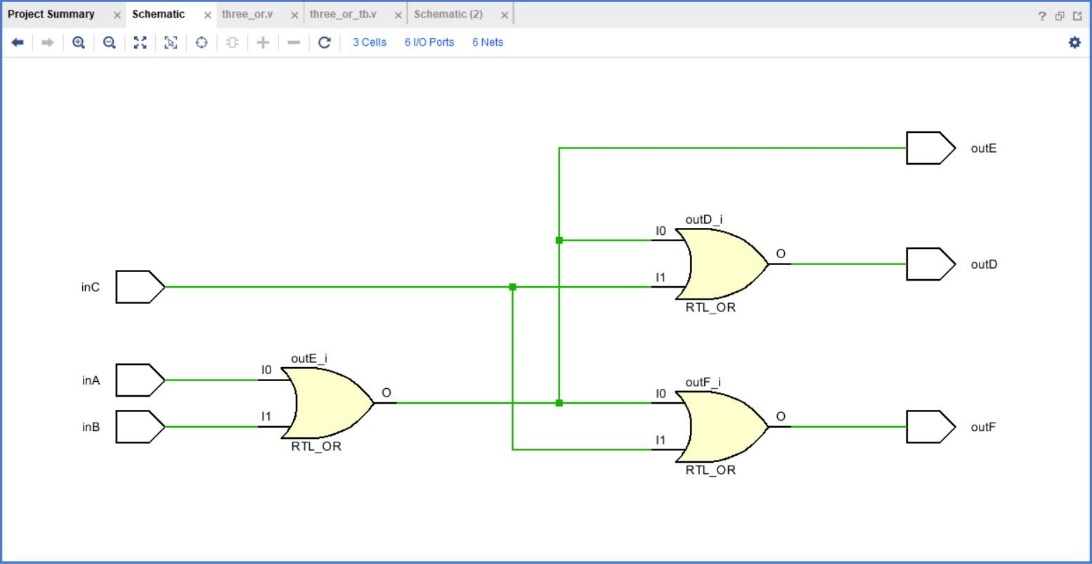
위의 <Figure 9>에서 (A)는 세 개의 입력이 모두 한 번에 하나의 OR gate에 인가되어 D라는 결과를 얻는 것이고, (B)는 AND gate에 A와 B, 두 개의 입력을 인가하여 D를 얻고, D와 C가 다른 OR gate의 입력으로 들어가 최종적으로 E라는 출력을 얻는 것입니다. 위에서 (A)의 출력 D는 outD로, (B)의 출력 D는 outE로, 출력 E는 outF로 설정하여 Verilog를 통해 구현해보면 다음 <Figure 10>와 같습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

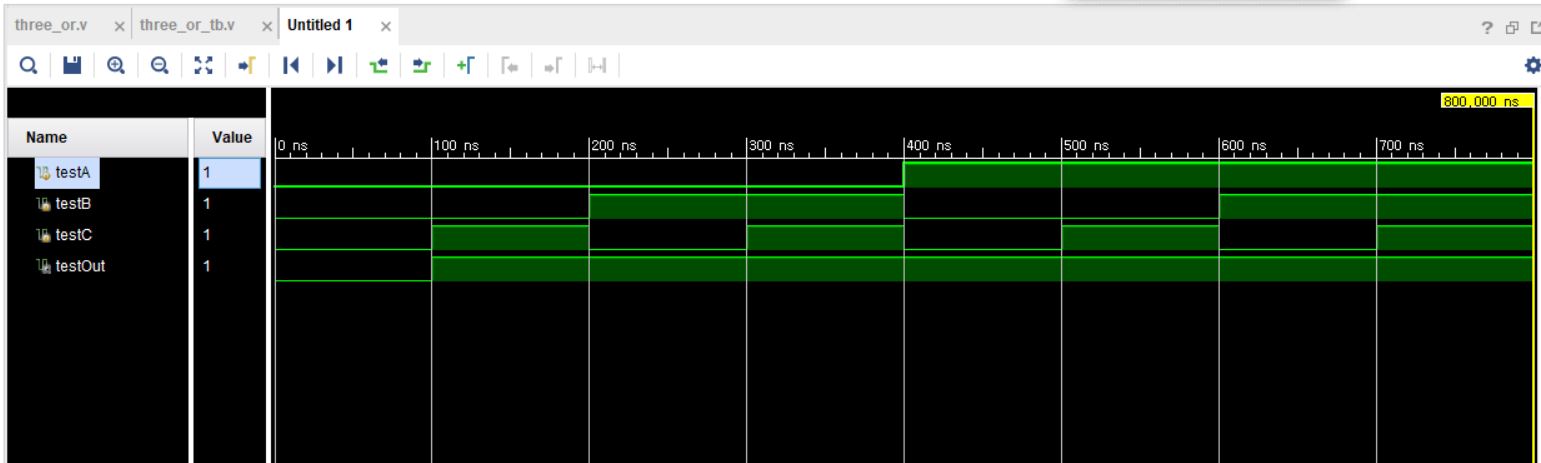
<Figure 10> three\_or.v 코드

<Figure 10>에 assign이 적힌 부분을 보면, outD는 inA와 inB, inC를 한 번에 이용하여 or 연산을 한 결과임을 알 수 있고, outE는 inA와 inB를, outF는 outE와 inC, 두 개의 입력을 각각 or 연산한 결과임을 알 수 있습니다. 이 코드의 schematic diagram을 확인해보면 다음 <Figure 11>와 같습니다.



<Figure 11> 3-input OR gate의 schematic diagram

위의 <Figure 11>를 보면, <Figure 9>에서 (A)와 (B)에서의 최종적인 출력, D와 E를 의미하는 outD를 얻는 과정이나 outF를 얻는 과정이 다르지 않음을 알 수 있습니다. <Figure 9>의 (A)처럼 3개의 입력이 하나의 OR gate로 인가되어 하나의 출력을 얻는다고 생각하고 <Figure 10>처럼 코드를 짜도 결국 <Figure 1-9>의 (B)와 동일하게 OR gate를 두 번 거치는 과정으로 구현된다는 사실을 알 수 있습니다.



<Figure 12> 3-input OR gate의 simulation 결과

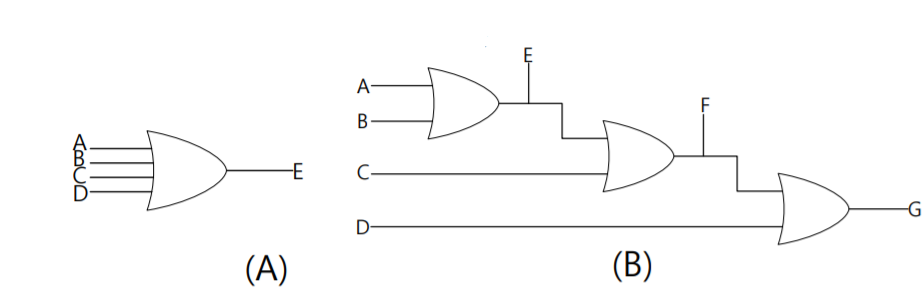
위의 <Figure 12>는 <Figure 11>을 통해 <Figure 9>의 (A)와 (B)는 동일한 출력 결과를 갖는다는 것을 확인한 후, test bench 코드를 이용하여 A, B, C라는 입력을 시간에 따라 바꿔가며 최종적인 출력 결과의 변화를 확인한 simulation 결과입니다 이 simulation 결과를 이용하여 3-input OR gate의 진리표를 작성해보면 다음 <Table 3>과 같습니다.

<Table 3> 3-input OR gate의 진리표

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | | | Output |
| A | B | C | D |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

OR 게이트는 입력 중 하나만이라도 1이면 1을 출력하므로 위의 <Figure 12>의 simulation 결과가 정확하게 나타난 것임을 알 수 있습니다. 이때, <Table 3>의 output D는 <Figure 9>의 (A)의 D와 (B)의 E를 의미합니다.

1. 4-input OR gate의 simulation 결과 및 과정에 대해서 설명하시오.



<Figure 13> 4-input OR gate를 구현하는 두 가지 방법

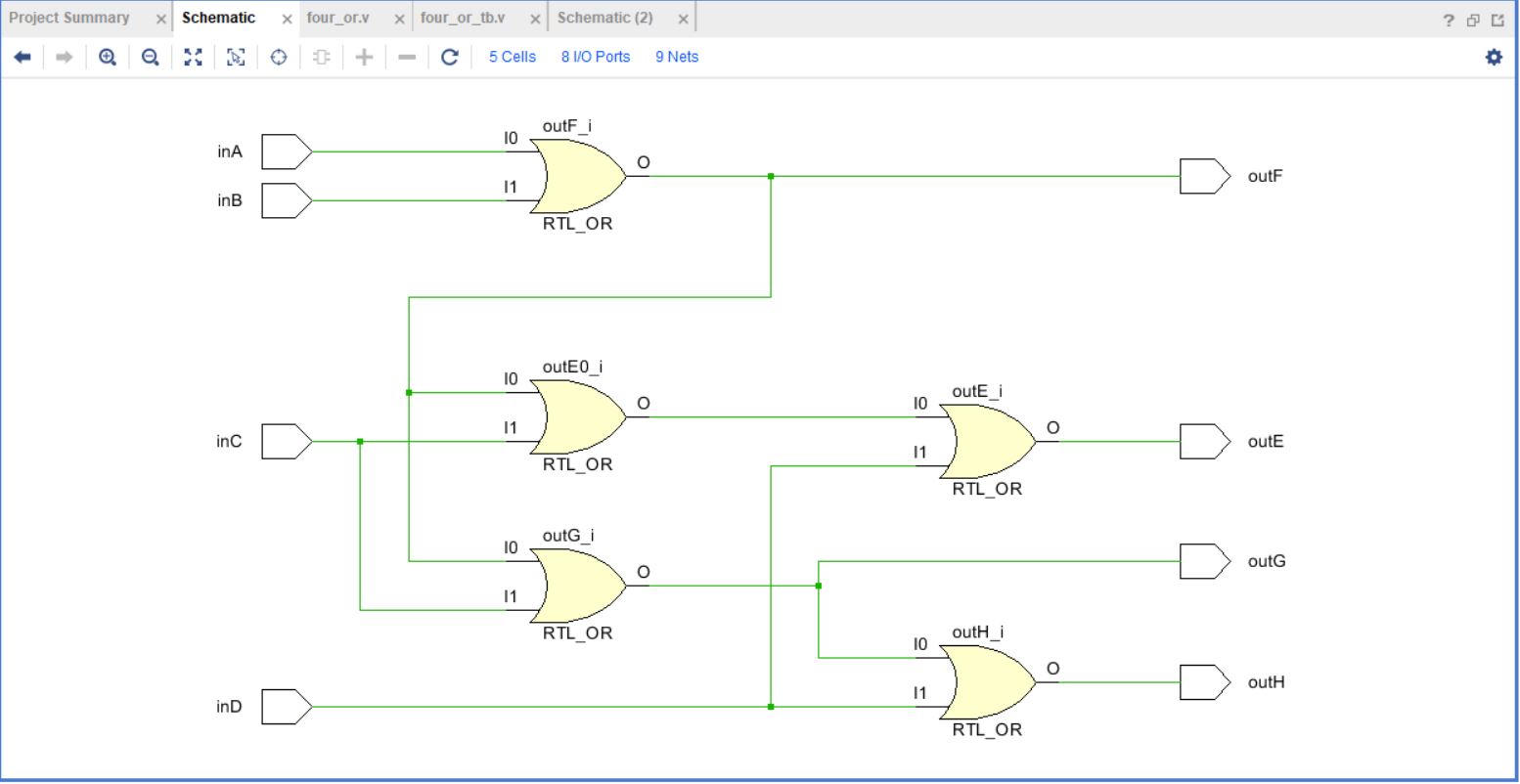
위의 <Figure 13>에서 (A)는 네 개의 입력이 모두 한 번에 하나의 OR gate에 인가되어 E라는 결과를 얻는 것이고, (B)는 OR gate에 A와 B, 두 개의 입력을 인가하여 E를 얻고, E와 C가 다른 OR gate의 입력으로 들어가 F를 얻고, F와 D가 다른 OR gate의 입력으로 들어가 최종적으로 E라는 출력을 얻는 것입니다. 위에서 (A)의 출력 E는 outE로, (B)의 출력 E는 outF로, 출력 F는 outG, 출력 G는 outH로 설정하여 Verilog를 통해 구현해보면 다음 <Figure 14>와 같습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

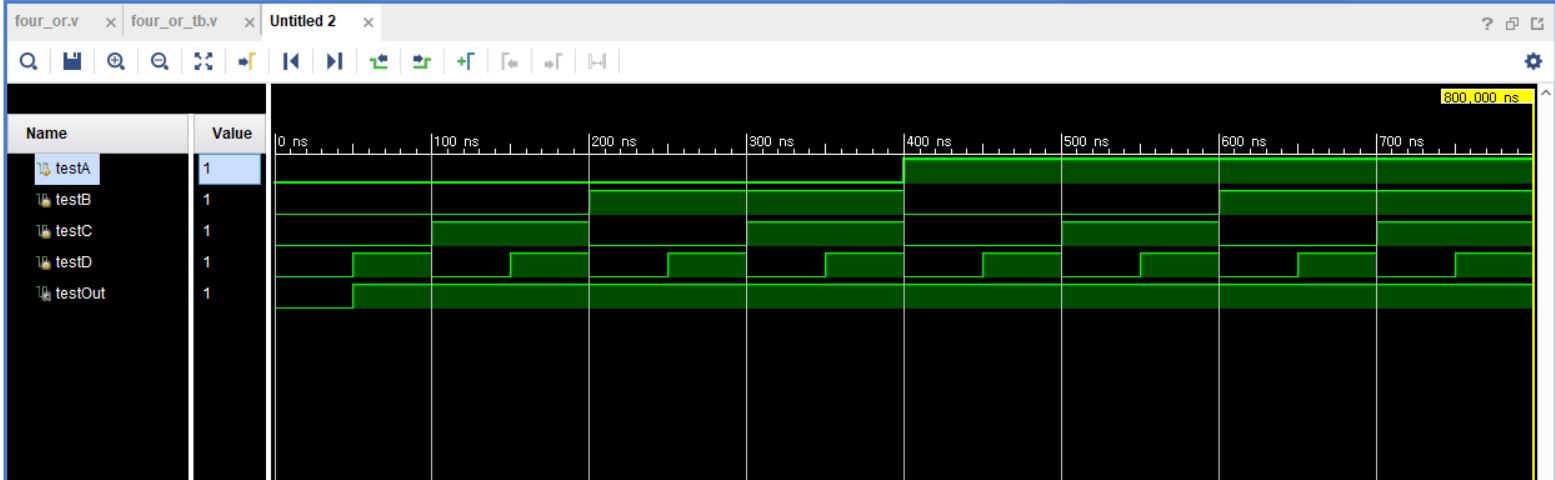
<Figure 14> four\_or.v 코드

<Figure 14>에 assign이 적힌 부분을 보면, outE는 inA와 inB, inC, inD를 한 번에 이용하여 and 연산을 한 결과임을 알 수 있고, outF는 inA와 inB를, outG는 outF와 inC를, outH는 outG와 inD를, 즉 두 개의 입력을 각각 and 연산한 결과임을 알 수 있습니다. 이 코드의 schematic diagram을 확인해보면 다음 <Figure 15>와 같습니다.



<Figure 15> 4-input OR gate의 schematic diagram

위의 <Figure 15>를 보면, <Figure 13>에서 (A)와 (B)에서의 최종적인 출력, E와 G를 의미하는 outE를 얻는 과정이나 outH를 얻는 과정이 다르지 않음을 알 수 있습니다. <Figure 13>의 (A)처럼 4개의 입력이 하나의 OR gate로 인가되어 하나의 출력을 얻는다고 생각하고 <Figure 14>처럼 코드를 짜도 결국 <Figure 13>의 (B)와 동일하게 OR gate를 세 번 거치는 과정으로 구현된다는 사실을 알 수 있습니다.



<Figure 16> 4-input AND gate의 simulation 결과

위의 <Figure 16>는 <Figure 15>을 통해 <Figure 13>의 (A)와 (B)는 동일한 출력 결과를 갖는다는 것을 확인한 후, test bench 코드를 이용하여 A, B, C, D라는 입력을 시간에 따라 바꿔가며 최종적인 출력 결과의 변화를 확인한 simulation 결과입니다 이 simulation 결과를 이용하여 4-input OR gate의 진리표를 작성해보면 다음 <Table 4>와 같습니다.

<Table 4> 4-input OR gate의 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inputs | | | | Output |
| A | B | C | D | E |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

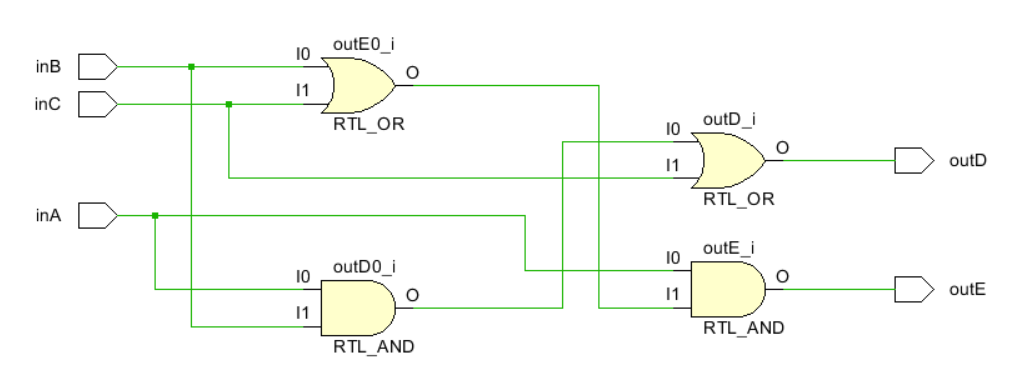
<Table 4>를 보면, OR 게이트는 입력의 개수와 무관하게 입력 중 하나만이라도 1이면 1을 출력하므로 위의 <Figure 16>의 simulation 결과가 정확하게 나타난 것임을 알 수 있습니다. 이때, <Table 4>의 output E는 <Figure 13>의 (A)의 E와 (B)의 G를 의미합니다.

1. 결과 검토 및 논의사항

본 실험을 통해, 입력 개수와 무관하게 AND 게이트는 모든 입력이 1이어야만 1을 출력하며 OR 게이트는 입력 중에 하나 이상이 1이면 1을 출력한다는 것을 실험적으로 확인할 수 있었습니다. 또한, 여러 개의 입력을 AND 연산이나, OR 연산으로 한 번에 계산하고 싶고 중간 연산 결과가 필요 없다면 <Figure 1>의 (A), <Figure 5>의 (A), <Figure 9>의 (A), <Figure 13>의 (A)처럼 식을 세워서 코딩을 해도 된다는 것을 알 수 있었습니다. 특정 입력의 연산 결과가 다른 곳에서도 필요할 때는 출력을 따로 설정하여 연산한다면 논리 게이트를 적게 사용할 수 있을 거라고 생각되었습니다.

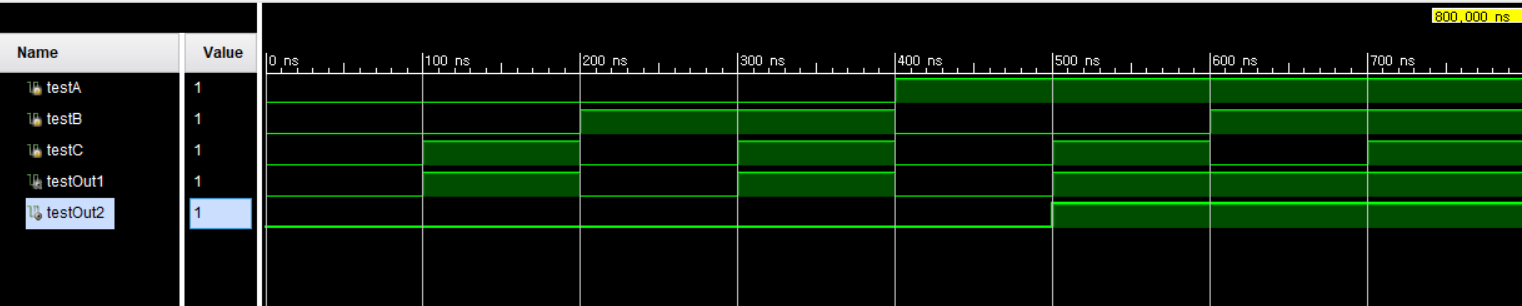
1. 추가 이론 조사 및 작성

본 실험은 입력 개수에 따른 AND 게이트와 OR 게이트 동작을 확인한 것으로, 입력 개수와 무관하게 두 개의 입력을 입력으로 하는 논리 게이트를 여러 번 거침으로써 최종 출력이 나타난다는 사실을 알 수 있었습니다. 본 실험에서는 어떤 입력부터 연산을 하는지가 크게 관련이 없었지만, AND 게이트와 OR 게이트가 섞인 Y=A&B|C와 같은 식에서는 어떠한 연산을 먼저 하는지에 따라 결과가 달라질 수 있기 때문에 D=(A&B)|C의 결과와 E=A&(B|C)의 결과를 확인하여 진리표를 만들어보고 비교해보면 좋을 거 같다는 생각이 들었습니다. 아래 <Figure a>는 D=(A&B)|C와 E=A&(B|C)의 schematic diagram입니다.



<Figure a> D=(A&B)|C와 E=A&(B|C)의 schematic diagram

inA, inB, inC(, inD) 순서대로 AND나 OR 게이트를 거쳤던 본 실험의 내용과 달리, 괄호를 어떻게 묶냐에 따라 연산 순서가 확연히 달라진 것을 확인할 수 있었습니다. 다음 <Figure b>은 test bench를 만들어 입력 A, B, C의 값에 따라 결과가 어떻게 달라지는지를 확인한 simulation 결과입니다.



<Figure b> D=(A&B)|C와 E=A&(B|C)의 simulation 결과

<Table a>는 위의 <Figure b>를 보고 진리표를 작성한 것으로 Out1은 위의 식에서 D를 의미하고, Out2는 위의 식에서 E를 의미합니다.

<Table a> <Figure b>의 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inputs | | | Output | |
| A | B | C | Out1 | Out2 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Out1과 Out2가 다른 것을 통해, AND 게이트와 OR 게이트가 섞여 있는 연산 식에서는 순서가 최종적인 출력 결과에 큰 영향을 미친다는 것을 확인할 수 있었고, 따라서 verilog를 이용해 코딩할 때 괄호를 적절하게 이용하여 식을 표현해야 한다는 것을 알 수 있었습니다.